MA3050 – INTRODUCTION TO LATTICE THEORY
Presentation Report

Group Members:
SIDHARTH UDAYAN (EP19BTECH11020)
SONA C VINOD (EP19BTECH11016)

C++ Program to perform certain operations in a Partially ordered set

With this program we can create a poset $(P, \leq)$ and:
a) check if $x \leq y \ \forall\ x,y \in P$
b) get Join and Meet of $(x,y) \ \forall\ x,y \in P$
c) get Maximal and Minimal elements in P.

Logic behind the program:

A poset is similar to a directed graph and so we use an adjacency list to represent graph G taking O(n) space, where n is the number of elements. First we give the number of elements n as input. Then elements of the poset are created, from 0 to n-1. We can then add edges i.e, $x_1 \leq y_1$ (adds an edge from x_1 to y_1), $x_2 \leq y_2$, … and so on.

To find if $x \leq y$, we need to check whether an edge exists from x to y. We do a BFS traversal starting at node x and check if y is reachable from x. BFS(x) returns the set of all nodes reachable from x, in a data structure known as vector which is available in the C++ Standard Template Library. BFS has a runtime complexity of $O(n + m)$, where m is the number of edges.

To find join of any two elements x and y,
(i) Take the intersection of BFS(x) and BFS(y). We use the set_intersection() method to store the intersection in another vector. Let $Z = BFS(x) \cap BFS(y)$.

(ii) We now determine $\forall$ node $k \in Z$, the number of nodes in Z that point to k. We do this by creating an array inBlack, where inBlack[k] stores the number of elements pointing to k. This step can be performed in $O(m + n)$ time complexity.

(iii) Then count the number of nodes with inBlack[k] = 0. The count() method in STL finds this in O(n) time. If there is only one such element, it is the join and return that element. If there is more than one such element in Z, then x and y probably has more than one upper bound element which are incomparable and so join does not exist. And one more, if Z = Φ**,** then x and y do not have any upper bound.

No how do we find meet of two elements? Keeping this in mind, at first we created another graph G^{rev} that would have all its edges in reverse manner as compared to G, and had been maintaining it. To find meet, follow the same procedure used to find join. This may be an inefficient strategy because it consumes just another space as G!

To find maximal elements in P, traverse through the adjacency list of G and return all the node/s whose linked list is empty.
To find minimal elements, traverse through the adjacency list of G^{rev} and return all nodes whose list is empty.

<u>Running the program:</u>

To run the program in Windows command prompt, first compile it by:
"g++ -o prog POSET.cpp" and press ENTER. Please note that .cpp file should be in the same location as the present location in command prompt.
Then run it by typing :
"prog"

To run the program in Linux system terminal, first compile it by:
"g++ -o prog POSET.cpp" and press ENTER. Please note that .cpp file should be in home directory.
Then run it by typing:
"./prog"

Or we can just directly compile and run if we are using an IDE.

Two screenshots have been attached which shows how program works.

Finally before program ends, it deletes all the stuff stored in heap memory to prevent possible memory leaks. Because heap is a particular type of memory allocation in which memory won't be freed unless we explicitly do so.